1 Objectives

Your goal in today's lecture is to gain some familiarity with OCaml. In particular, you should know...

- how to use immediate mode for interactive programming.
- the basic builtin types.
- how to use pattern matching in your functions.
- how to use tuples and lists.
- know the four ways to create variables, and how long they last.
- that **whenever you see a function, you always ask "what's its type?".**

2 Problems

Here are some example problems which will be useful to study in preparation for the exam. Some of these may be done in class.

1. Consider this code:

```
let x = 27;;
let foo x =
   let x = 5 in
     (fun x -> print_int x) 10;;
foo 12;;
```

   What value will be printed?

   - a) 5
   - b) 10
   - c) 12
   - d) 27

2. Consider this code:

```
let x = 27;;
let foo x =
   let x = 5 in
     (fun y -> print_int x) 10;;
foo 12;;
```

   What value will be printed?

   - a) 5
   - b) 10
   - c) 12
   - d) 27

3. Consider this code:

```
let x = 27;;
let foo x =
   let y = 5 in
     (fun y -> print_int x) 10;;
foo 12;;
```

What value will be printed?

- a) 5
- b) 10
- c) 12
- d) 27

4. Consider this code:

```
let x = 27;;
let foo y =
   let y = 5 in
     (fun y -> print_int x) 10;;
foo 12;;
```

What value will be printed?

- a) 5
- b) 10
- c) 12
- d) 27

5. What will be the output of the following code?

```
let x = 20;;
let f = fun x -> x + 1;;
let y = f 30;;
print_int x;;
```

6. One of the lists below is invalid. Which one?

- a) `[2; 3; 4; 6]`
- b) `[2,3; 4,5; 6,7]`
- c) `[2.3,4; 3.2,5; 6,7.2]`
- d) `[["hi"; "there"]; ["how"]; []; ["goezit"]]`

7. What is the type of the following function:

```
let f x = x + 1
```

8. What is the type of the following function:

```
let f x = x +. 1.0
```

9. What is the type of the following function:

```
let f x = x :: [1]
```

10. What is the type of the following function:

```
let f x = x @ [1]
```

11. What is the type of the following function:

```
let f x = 1 :: x
```

12. Write an OCaml function that inspects a list. If the list is empty, output `"empty"`, otherwise, output `"not empty"`.

13. Write an OCaml function that returns the first element of a list. Assume that only non-empty lists will be given to the function.

14. Write an OCaml function that returns the second element of a list. Assume that only non-empty lists will be given to the function.

15. Write an OCaml function that returns the sum of the first three elements of a list. The list may have any number of elements. Do not use recursion, just use `match/with`.